Using the Functional Mockup Interface as an Intermediate Format in AUTOSAR Software Component Development

Bernhard Thiele[*], Dan Henriksson[+]
[*]German Aerospace Centre (DLR), Institute for Robotics and Mechatronics, Germany
[+]Dassault Systèmes AB, Ideon Science Park, Lund, Sweden
Bernhard.Thiele@dlr.de, Dan.Henriksson@3ds.com

Abstract

This paper shows how the recently developed Functional Mockup Interface (FMI) standard for model exchange can be utilized in the context of AUTOSAR software component (SW-C) development. Automatic transformations between the XML schemas of the two standards are utilized to convert FMI models to AUTOSAR. An application example is demonstrated, where a Modelica controller is exported through FMI, converted to an AUTOSAR SW-C and then imported into an AUTOSAR tool. The presented approach, with FMI as an intermediate format, should be an attractive alternative to providing full-fledged AUTOSAR SW-C export.

Keywords: FMI; AUTOSAR; model-based design; embedded software

1 Introduction

During the last two years, an open standard for exchange of simulation models, the Functional Mock-Up Interface (FMI), has been developed within the European ITEA2 research project MODELISAR. This standardized interface supports exchange of models that are described by differential, algebraic and discrete equations with time-, state- and step-events. The first official version, 1.0, of this standard was released on January 26, 2010.

Apart from the obvious improvements for model exchange between different tools and vendors, the interface is also well suited, and designed, for software components in embedded control systems. Since one of the major industrial driving forces behind the MODELISAR project is within the automotive industry, interoperability of the lightweight FMI with the comprehensive AUTOSAR standard for automotive E/E applications is of high interest. This paper examines the applicability of using FMI within an AUTOSAR-based software component development process.

The paper is organized as follows. Details of the FMI and AUTOSAR standards are given in Sections 2 and 3, respectively. A mapping and conversion between FMI and AUTOSAR is then described in Section 4. An example application involving the Dymola [1] and AUTOSAR Builder [2] tools are presented in Section 5. Finally, Section 6 gives the conclusions.

2 Functional Mockup Interface

Integration of components delivered by many different suppliers is a common task in modern product engineering. To reduce costs, control complexity, and accelerate development it is desirable to allow this integration task to be done using a virtual representation of the product, i.e., to build a digital mock-up. Besides spatial integration of the different components in a CAD tool, it is also required to let the dynamic behavior of the product to be predicted and checked by means of (physical) simulation.

Very often suppliers already have dynamic system models of their particular component, developed within their preferred simulation tool. However, integrating the various component models (possibly each developed with a different simulation tool) into an overall system model for joint simulation has proven to be a rather difficult, time-consuming, and numerically fragile undertaking.

The intention of the Functional Mockup Interface (FMI) is that dynamic system models from different tool vendors can be coupled together to form an overall system model with minimal effort and high numerical quality. To achieve that goal, the FMI defines an open interface that needs to be implemented by tools in order to import or export FMI system models. In FMI terminology a system model that implements the interface defined by the FMI specification is called a Functional Mockup Unit (FMU).

Figure 1 from the MODELISAR project profile description shows a use-case from an automotive OEMs perspective.

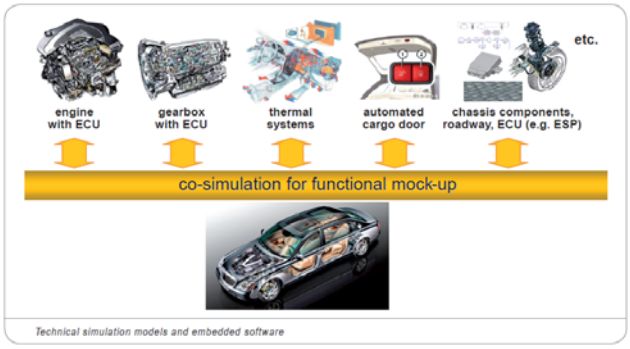

Figure 1: A functional mock-up of a vehicle consisting of several coupled Functional Mockup Units (source: www.itea2.org)

3 Introduction to AUTOSAR

AUTOSAR is an automotive standard, which aims to decouple hardware and software and to separate communication from function. It achieves that by introducing several layers of abstraction with standardized interfaces.

The development partnership AUTOSAR (http://www.autosar.org) has released version 4.0 of the AUTOSAR standard in December 2009. However, since most commercially available tools to this date not yet support the 4.0 release, the following discussion concentrates on the 3.1 release of the standard.

The actual functional behavior (e.g. a model-based control algorithm) is encapsulated in AUTOSAR Software Components (SW-Cs). These components are decoupled through standardized interfaces from specific characteristics of Electronic Control Units (ECUs) and the given communication mechanism (e.g., automotive buses like CAN, FlexRay, LIN or inter-process communication if several software components interact on the same ECU).

The benefit of this decoupling is that the software components can be moved without adaption between different ECUs. The interconnections between the software components are handled by the Virtual Functional Bus (VFB). The VFB is the sum of all communication mechanisms and essential interfaces to the basic (hardware-dependent) software provided by AUTOSAR on an abstract level to software components (see Figure 2).

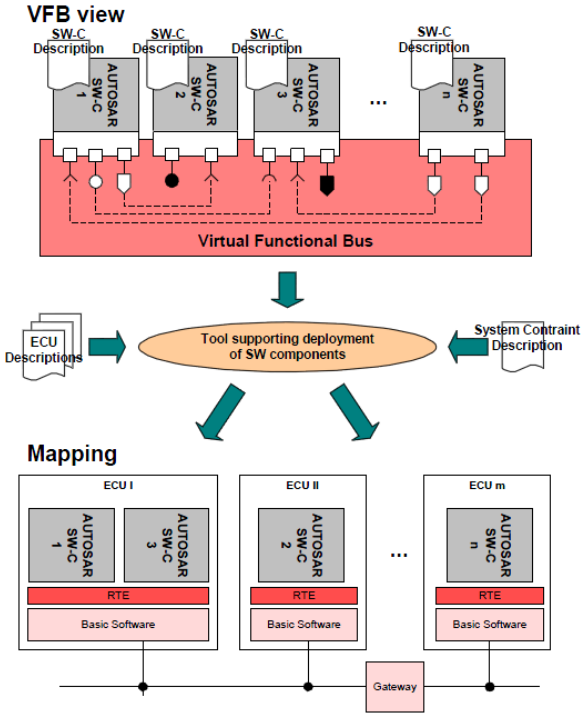

Figure 2: Basic AUTOSAR approach for configuration of an AUTOSAR system (source: [7], p. 9).

The mapping of the software components to the physical ECUs, as well as the mapping of the software component's communication ports to the physical communication mechanism (e.g., CAN, FlexRay, LIN, or shared memory) is provided in a later configuration step. This allows starting the development of the logical software functions independently from the decision of the target platform (following the concept of separation of *logical system architecture* from the *technical system architecture* [3] [4]).

After that configuration, an AUTOSAR tool can deduce what software/communication functionality is required on a particular ECU and will be able to generate the needed source code for the particular ECU (target platform). This means that the abstract communication connections modeled on the VFB level are transformed to concrete communication connections on the ECUs. The software layer that provides the VFB communication services for the SW-C is called AUTOSAR Runtime Environment (RTE) and needs to be generated by the tool for every ECU.

4 FMI to AUTOSAR Software Component conversion

The development of the FMI is primarily intended to provide a standardized exchange format for *physical*

simulation models [1]. Nevertheless the intention to use that standard also for software components in embedded control systems is already stated in the abstract of [6].

Compared to AUTOSAR, the FMI standard is much smaller and more straightforward, and support of the FMI standard is a more manageable task[1]. Thus, a conversion from FMI to AUTOSAR SW-Cs could be a cost effective alternative to providing dedicated AUTOSAR code generators (especially if support for FMI is already available or planned).

4.1 Establishing a relation between FMI and AUTOSAR software component specification methodology

Both FMI and AUTOSAR use XML documents for capturing the information about the (software) model (see [6] and [7]). In each case, the structure of the XML documents is defined in an associated XML schema [8]. A notable difference is that the AUTOSAR 3.1 schema occupies about 1000KB, while the FMI 1.0 schema is limited to about 25.5KB.

Mapping between different XML schemas is a common IT task and dedicated standards and tools are readily available. The Altova MapForce [9] program is a tool that allows defining mappings between XML schemas in a graphical manner. Figure 3 shows an excerpt of a mapping from FMI to AUTOSAR 3.1 developed in MapForce which was utilized in the first prototype mapping[2].

There is no univocal relation between FMI and AUTOSAR elements. Therefore design decisions about the available alternatives need to be made.

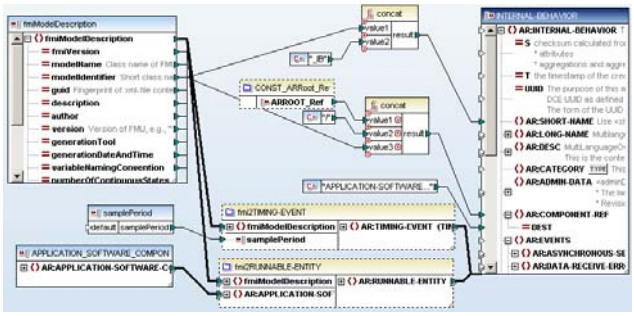

Figure 3: Excerpt of the mapping between the FMI and AUTOSAR schema.

4.2 Mapping FMI inputs/outputs to AUTOSAR SW-Cs Ports

The interaction between AUTOSAR Software Components and other parts of the system (including other AUTOSAR Software Components) is realized over a set of ports with standardized interfaces. Figure 4 shows the graphical representation of an AUTOSAR SW-C with different ports at its interface boundary.

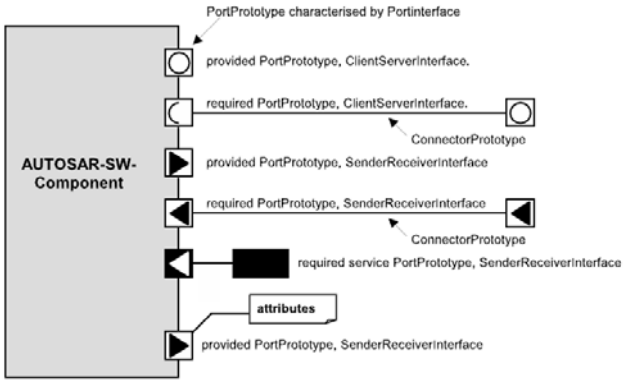

Figure 4: Graphical representation of software-components in AUTOSAR (source: [12], p. 20).

There are basically three kinds of port interfaces supported by AUTOSAR:

- Client-server: The server is the provider of operations and several clients can invoke those operations.
- Sender-receiver: A sender distributes information to one or several receivers, or one receiver gets information (events) from several senders.
- Calibration: Using or providing (static) calibration data

A port can either be a "*PPort*" or an "*RPort*". A "*PPort*" provides the elements defined in a port interface. An "*RPort*" requires the elements defined in a port interface.

The FMI standard collects all visible/accessible variables within one central data structure (in the "*ModelVariables*" element). That element contains a

[1] In particular the import of AUTOSAR SW-Cs is much more complex, than that of importing an FMU. The reason for this is the great flexibility of the AUTOSAR standard to define SW-Cs, which needs to be managed by an importer. So using FMI as interchange format for embedded software components could also facilitate the exchange of embedded software.

[2] In later versions the mapping in MapForce was dropped in favor of a mapping developed in Scala [10] and Java utilizing auto-generated XML data bindings from the Altova XMLSpy tool [11]. The reason for that was the perceived need for more flexible language expressiveness as the mapping became more complex.

sequence of elements of the type "*fmiScalarVariable*" as shown in Figure 5.

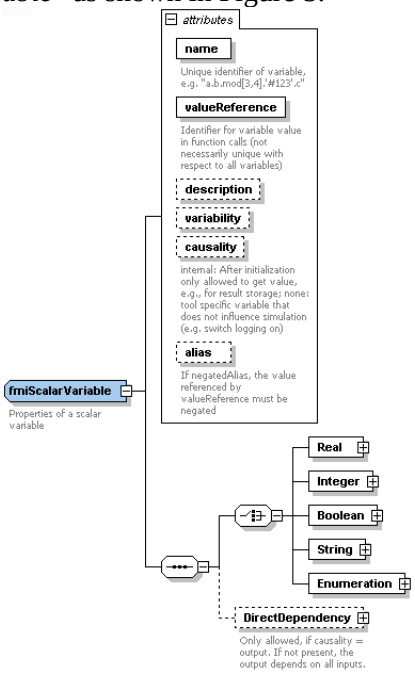

Figure 5: Structure of the fmiScalarVariable element

The information whether a variable is an input or output to the component (and therefore interface relevant) is coded in the optional attribute "*causality*" (condition: "*causality = input/output*"). Parameters for potential calibration of the component are identified through the "*variability*" attribute (condition: "*variablity = parameter*"). FMI inputs/outputs map to the AUTOSAR sender-receiver port interface (input maps to "*RPort*" and output maps to "*PPort*").

AUTOSAR supports different flavors of sender-receiver port communication (explicit/implicit communication, queued or un-queued communication, sending/receiving of data or events). There is no counterpart for these options in the FMI standard. Consequently, the desired mapping needs to be decided at the FMU import. For the further discussion we assume that FMI inputs/outputs are mapped to explicit, un-queued, data communication ports.

4.3 Mapping of FMI parameters to AUTOSAR calibration ports

In FMI a parameter is identified by the condition "*variablity = parameter*" within the variable definition (see above). Parameters can be set before initializing the FMU. After initialization they are fixed and may not change during runtime.

In embedded automotive software design, manipulation of parameters is termed calibration. AUTOSAR provides flexible support for manipulating calibration parameters.

- Port-based calibration: Parameters are explicitly visible on the VFB. This mechanism is meant for public parameters of a SW-C (e.g. in Figure 8 the parameters for the PI-controller are port-based, public parameters).
- Private calibration parameters: These reside internally within a SW-C. They are not explicitly visible on the VFB level.

The rationale for differentiating between "private" and "public" parameters is that a supplier might want to indicate which parameters are safe to be calibrated by the OEM and which parameters the OEM should better not touch. Additionally, AUTOSAR allows to specify whether parameters may be calibrated online (while the software function is running), or only before initialization.

Like the previous mapping of FMI inputs/outputs to AUTOSAR ports, there is no univocal mapping from FMI parameters to AUTOSAR calibration parameters. However, it seems to be reasonable to map FMI parameters to "public" calibration ports, explicitly visible at VFB level[3].

4.4 Wrapping the FMU C-code into an AUTOSAR Runnable Entity

Through its ports, the AUTOSAR SW-C specifies which information it requires from and provides to other components. The actual implementation of a component consists of a set of "runnable entities" (in short runnable[4]), which are code sequences in the SW-Cs that are activated through events, like timers or the receiving of data.

In order to execute an FMU as an AUTOSAR SW-C, it is necessary to wrap the C-function calls to the FMU into an AUTOSAR runnable.

Every runnable entity provides an entry point and an associated set of data. For components implemented using C or C++ the entry point of a runnable is implemented by a function with global scope defined in the source code of the software component. The RTE is the sole entity that can trigger the execu-

[3] The current limitation of the FMI standard to allow parameters only to be set before initialization is in contrast to the well-established practice of online-calibration of controller algorithm parameters. Hopefully, future versions of the FMI standard will deal with that limitation.

[4] A runnable runs in the context of a task. The task provides the common resources to the runnables such as context and stack-space. On the operating system level a task can be realized as either a full process or as a light-weight thread.

tion of a runnable. In [13], p.141 the signature of this function is defined as

<void|Std_ReturnType> <name>([IN RTE_Instance <instance>], [role parameters])

AUTOSAR provides various events that can trigger a runnable (e.g. TimingEvent, DataReceivedEvent, DataReceiveErrorEvent, DataSendCompletedEvent, etc.). For using Modelica/FMU controller models in AUTOSAR applications the cyclic invocation plays the most important role. For that purpose the TimingEvent is used as activation method for FMU models.

Since the AUTOSAR activation of runnables is targeted at discrete controllers it does not support the concept of a solver, which is of course needed in the FMI specification. As a consequence, an adequate FMI solver must be wrapped inside the runnable functions. A design decision is needed whether FMUs with continuous states ("*numberOfContinuousStates* > *0*") shall be supported by the AUTOSAR importer, or if the import is restricted to purely discrete FMUs (superseding the need of wrapping a numerical integrator into the runnable). For the purpose of this work it is decided to only allow purely discrete FMUs[5].

Notably, FMI 1.0 does not include an attribute for specifying a fixed sample period[6]. Thus, the sample period for the TimingEvent needs to be given as a parameter within the FMU import process.

5 Example application

The FMI to AUTOSAR conversion will be demonstrated in an application example. In this scenario we will consider export of a Modelica controller from Dymola through FMI. The exported FMU will then be converted to AUTOSAR and imported into the AUTOSAR Builder tool.

In order to focus the discussion, a simple, instructive example of a controlled drive is used. The example is modeled in Modelica using the Dymola tool (see Figure 6). The reference trajectory is provided by the "*reference*" block. The "*pIController*" block implements the closed-loop control of the plant.

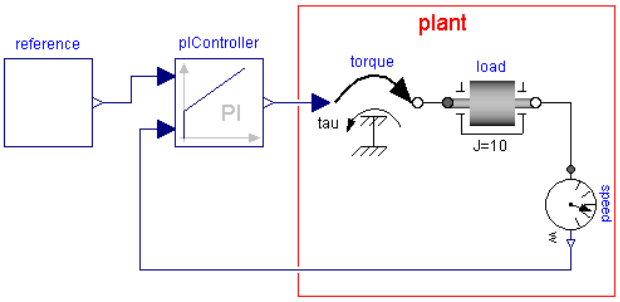

Figure 6: Simple controlled drive example as Modelica model in Dymola

The PI-controller (proportional-integral controller) may be parameterized with the proportional gain "*k*" and the time constant "*T*" of the integral term, as shown in Figure 7.

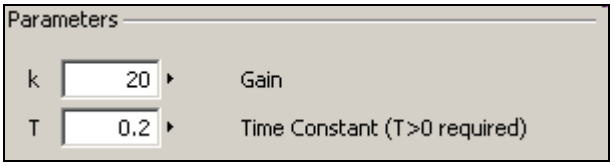

Figure 7: Parameter dialog for controller calibration in Dymola

Figure 8 shows how the example can be modeled within an AUTOSAR VFB diagram. The parameters are modeled as explicit inputs to the "*PIController*" SW-C. The "PI_Init" runnable initializes the controller and sets the provided parameters. The "PI_Run" runnable is called periodically to provide the required actuating variable. Instead of the plant, Sensor-Actuator SW-Cs have been introduced ("*TorqueActuator*" and "*SpeedSensor*").

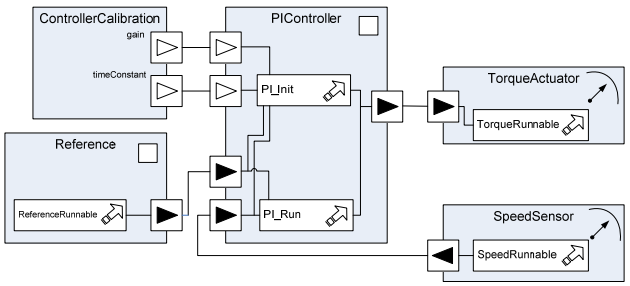

Figure 8: Simple controlled drive example as AUTOSAR VFB diagram (including sensor and actuator components, as well as parameter ports for controller calibration)

The Modelica PI-controller is exported from Dymola as an FMU and transformed from the FMI schema to the AUTOSAR schema. Similarly, the required C wrapper code for the AUTOSAR runnable is automatically generated from the FMI schema. In both cases Scala and Java are used as the implementation languages of choice for carrying out

[5] This restriction is not as severe as it may seem on first sight. If it is desired to use models with continuous states, some tools (e.g. Dymola) provide options of exporting such models as FMUs with inline integrators. As a result the exported FMU has no external continuous states ("*numberOfContinuousStates* = *0*"), thus no integrator needs to be provided for executing such an FMU.

[6] Hopefully, future versions of the standard will allow specifying a fixed sample period.

these transformations[7]. In Figure 9 an excerpt of the Model Description File of the PI-controller as exported by Dymola is given.

```
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription
  fmiVersion="1.0"
  modelName="PI"
  modelIdentifier="PI"
  guid="{8362d077-2c00-4f74-975b-339843f12ce7}"
  variableNamingConvention="structured"
  numberOfContinuousStates="0"
  numberOfEventIndicators="0">
  <ModelVariables>
    <ScalarVariable
      name="k"
      valueReference="16777216"
      description="Gain"
      variability="parameter">
      <Real start="20"
        fixed="true"/>
    </ScalarVariable>
    <ScalarVariable...
    <ScalarVariable...
    <ScalarVariable
      name="u_ref"
      valueReference="352321536"
      causality="input">
      <Real/>
    </ScalarVariable>
    <ScalarVariable...
    <ScalarVariable
      name="y"
      valueReference="335544320"
      causality="output">
      <Real/>
    </ScalarVariable>
    <ScalarVariable...
    <ScalarVariable...
  </ModelVariables>
</fmiModelDescription>
```

Figure 9: Excerpt from the Model Description File of the PI-controller (FMI schema compliant xml format)

The necessary workflow for transforming the FMU to an AUTOSAR SW-C is depicted in Figure 10. The workflow is highly automated, since the current version of the fmi2autosar program needs no user interaction except of specifying the location of the program's input and the desired fixed sample period. Basically, the import into AUTOSAR Builder works by just copying the files generated by fmi2autosar into an AUTOSAR project directory and "refreshing" the project[8]. The screenshot in Figure 11 shows the AUTOSAR Master Editor view, after the PI-controller import.

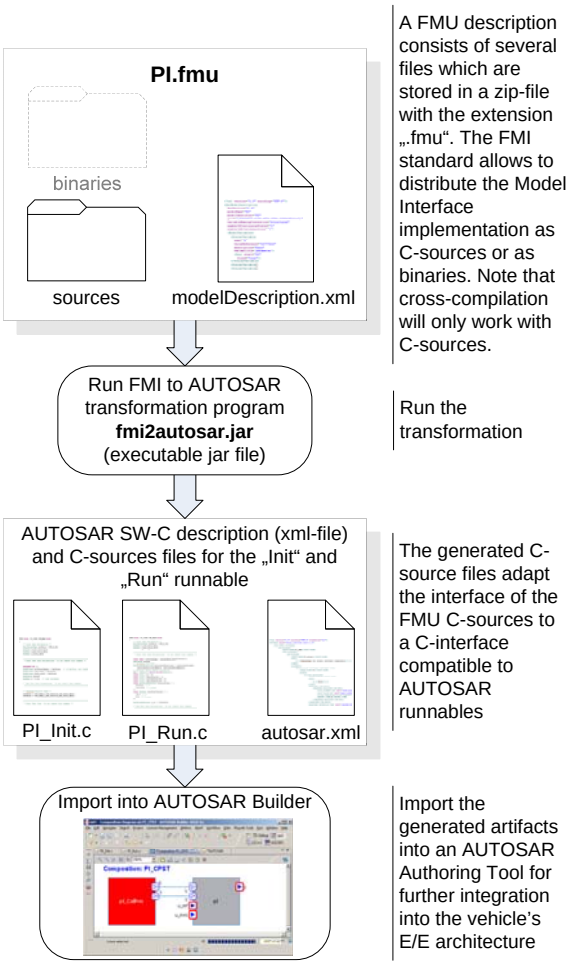

Figure 10: FMU to AUTOSAR-SW-C transformation workflow demonstrated through the PI-controller example

AUTOSAR allows a flexible structuring of elements through the use of packages and subpackages. To achieve a well-arranged layout, which facilitates integration into an AUTOSAR project, the proposed transformation collects all elements resulting from an FMU transformation into one package. The value of the FMU's *"modelIdentifier"* attribute is used as base string for the package and subpackage names (see Figure 11).

After the import the model can be further processed in AUTOSAR Builder. It can be integrated with other SW-Cs and simulated on the VFB level using the Geensoft ASim tool.

6 Conclusions

This paper has presented a mapping and conversion scheme between the Functional Mockup Interface (FMI) for model exchange and the automotive software architecture standard, AUTOSAR. A suitable subset of the AUTOSAR software component speci-

[7] The implementation effort was considerably reduced by leveraging the functionality of the XMLSpy tool [11] to automatically generate XML data bindings for the Java language.

[8] For further processing in AUTOSAR Builder, e.g., simulation on VFB level and RTE generation, necessary build dependencies and compiler flags need to be configured manually in AUTOSAR Builder. Because the required settings are highly tool- and application-specific no attempt is made to provide default settings.

fication was selected for the mapping and the rationale for these decisions was motivated. The design has been validated by importing the transformed FMI models into an AUTOSAR Authoring Tool and simulating the design on the Virtual Functional Bus level.

The FMI to AUTOSAR mapping process has also identified missing features in FMI that should be worth considering for future versions of the standard.

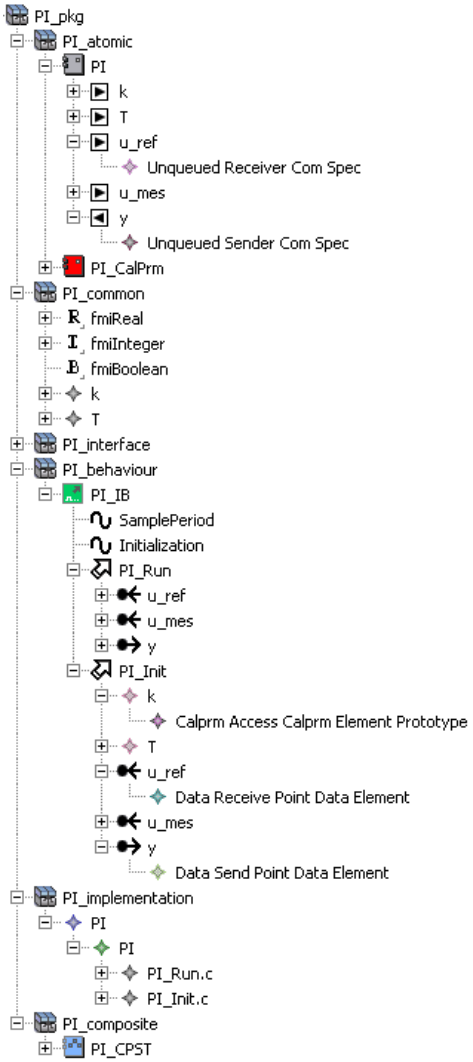

Figure 11: Screenshot of the AUTOSAR Master Editor after importing the FMU of the PI controller into the Geensoft AUTOSAR Builder tool

6.1 Acknowledgements

Partial financial support of DLR by BMBF (BMBF Förderkennzeichen: 01lS08002) for this work within the ITEA2 project MODELISAR is highly appreciated.

Dassault Systèmes AB thanks the Swedish funding agency VINNOVA for partial funding of this work within the ITEA2 project MODELISAR (2008-02291).

References

[1] Dymola Version 7.4. Dassault Systèmes, Lund, Sweden (Dynasim). Homepage: http://www.dymola.com, 2010

[2] AUTOSAR Builder Version 2010-2a. Dassault Systèmes, Brest, France (Geensoft). Homepage: http://www.geensoft.com/, 2010

[3] Jörg Schäuffele, Thomas Zurawka. Automotive Software Engineering, SAE International, 2005

[4] Hilding Elmqvist, Martin Otter, Dan Henriksson, Bernhard Thiele, Sven Erik Mattsson. Modelica for Embeeded Systems, 7th Int. Modelica Conference, 2009

[5] Barbara Lange. Verbunden; Austauschformat für die Simulation, iX extra 10/2010, p. VIII

[6] MODELISAR consortium. Functional Mock-up Interface for Model Exchange 1.0, http://modelisar.org/specifications/FMI_for_ModelExchange_v1.0.pdf, 2010

[7] AUTOSAR GbR. Technical Overview, AUTOSAR, Part of Release 3.1, AUTOSAR_TechnicalOverview.pdf, 2008

[8] W3C. XML Schema Part 1: Structures Second Edition. W3C Recommendation 28 October 2004, http://www.w3.org/TR/xmlschema-1/

[9] MapForce 2010, Altova GmbH, Vienna, Austria, http://www.altova.com/mapforce.html, 2010

[10] Martin Odersky. The Scala Language Specification Version 2.8, http://www.scala-lang.org/docu/files/ScalaReference.pdf, 9 November 2010

[11] XMLSpy 2010, Altova GmbH, Vienna, Austria, http://www.altova.com/xmlspy.html, 2010

[12] AUTOSAR GbR. Software Component Template, Part of Release 3.1, AUTOSAR_SoftwareComponentTemplate.pdf, 2010

[13] AUTOSAR GbR. Specification of RTE, Part of Release 3.1, AUTOSAR_SRS_RTE.pdf, 27.01.2010